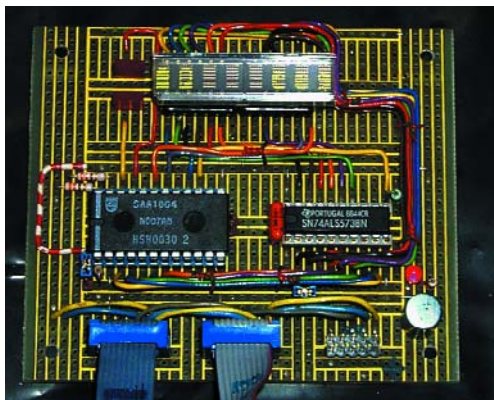


I²C General Purpose Driver Interface

with a practical application

By W. Huiskamp

The I²C interface has become a popular way to hook up external devices to microprocessors. However popular the I²C bus may be, there are still many peripherals that use a parallel interface. This article presents a low-cost I²C port expander that can be used to control peripherals without an I²C interface. Our example shows how to connect the General Purpose Driver Interface to an alphanumeric LED display.



Note: The circuit and software described in this article have not been tested in the Elektor Electronics design laboratory.

I²C slave devices range from simple input/output ports to complex parts like EEPROMs. Devices without an I²C interface can be connected to the I²C bus by means of a Slave microprocessor with I²C hardware. This solution makes the slave 'smarter' and takes some load off the main processor but the disadvantages are that a Slave microprocessor adds cost to the system while software needs

to be developed for both the Master and Slave processor. I²C port expanders are a good alternative to interface peripherals to the I²C bus.

Hardware Concept

I²C port expanders translate I²C messages into 8 bits of parallel data. The best known I²C port expander is the PCF8574. This device can control slaves like character LCD's using the '4 bit' mode of the HD44780 [1]. However, many slave devices need a wider interface: 8 bit databus, some address bits and several control lines. Two or more PCF8574's could do the trick, but it quickly results in costly hardware and unwieldy software. Our design employs the SAA1064 device, which has two 8 bit ports capable of driving LEDs directly (see **Figure 1**).

The SAA1064 is normally used in multiplex mode, allowing direct drive of 32 LEDs (see [2]). The device can also operate in a non-

multiplexed (static) mode resulting in 16 available output pins. The 'I²C General Purpose Driver Interface' employs the static mode to provide an 8-bit 'address/databus' and an 8-bit 'control bus'. In a way, the design emulates the multiplexed Address / Data bus and Control bus of an 8051 processor. There is one restriction: the SAA1064 only supports 'outputs'. The Data bus can not be used to read back from the Slave device. In many cases, this limitation is acceptable for simple slaves.

Hardware Design

Figure 2 shows the schematic diagram of the Driver Interface implemented for a Hewlett-Packard (now Agilent [3]) 'Smart Alphanumeric Display'. A brief description of the design is given below.

The I²C bus connector provides SDA, SCL, GND and regulated 5 V power. Two I²C connectors are used to allow daisy chaining. SDA and SCL lines are connected through a

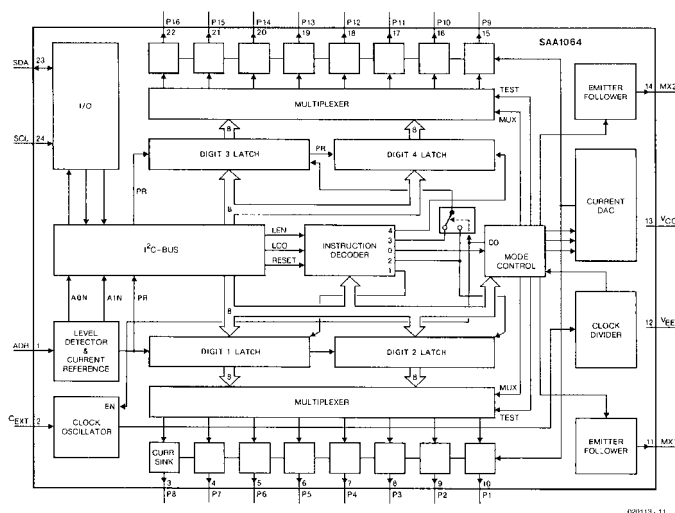


Figure 1. SAA1064 block diagram.

series resistor (R13, R14) to afford protection against too high voltages. The I²C pull-up resistors for SDA and SCL must be provided externally, e.g., on the I²C Master circuit board.

The SAA1064 device (U1) receives and decodes I²C traffic and provides

the address/data on Port 1 (P1-P8) and control signals on Port 2 (P9-P16). The SAA1064 can be assigned to one of four possible I²C slave addresses. The voltage on pin 1 of U1, as set by jumpers J1-J3, selects the active address :

J1	J2	J3	Slave Address
—	—	x	0111000R/W
—	—	—	0111001R/W
—	x	—	0111010R/W
x	—	—	0111011R/W

Where 'x' denotes the presence of a jumper.
R/W indicates the lowest address bit.

So, for example:

01110000 = 70H = write

01110001 = 71H = read

Note that only one jumper is allowed. Fitting both J1 and J3 will short out the power supply !

Port 1 of U1 emulates the multiplexed Address/Databus. The pins are connected to the Databus of the slave device and to the SN74ALS573 latch (U2). The latch freezes the Address bits on the falling edge of the pulse at the ALE (Address Latch Enable) pin. The latch output is the Address bus for the slave device.

Port 2 of U1 emulates the 'Control bus' for the slave device. It provides the following signals:

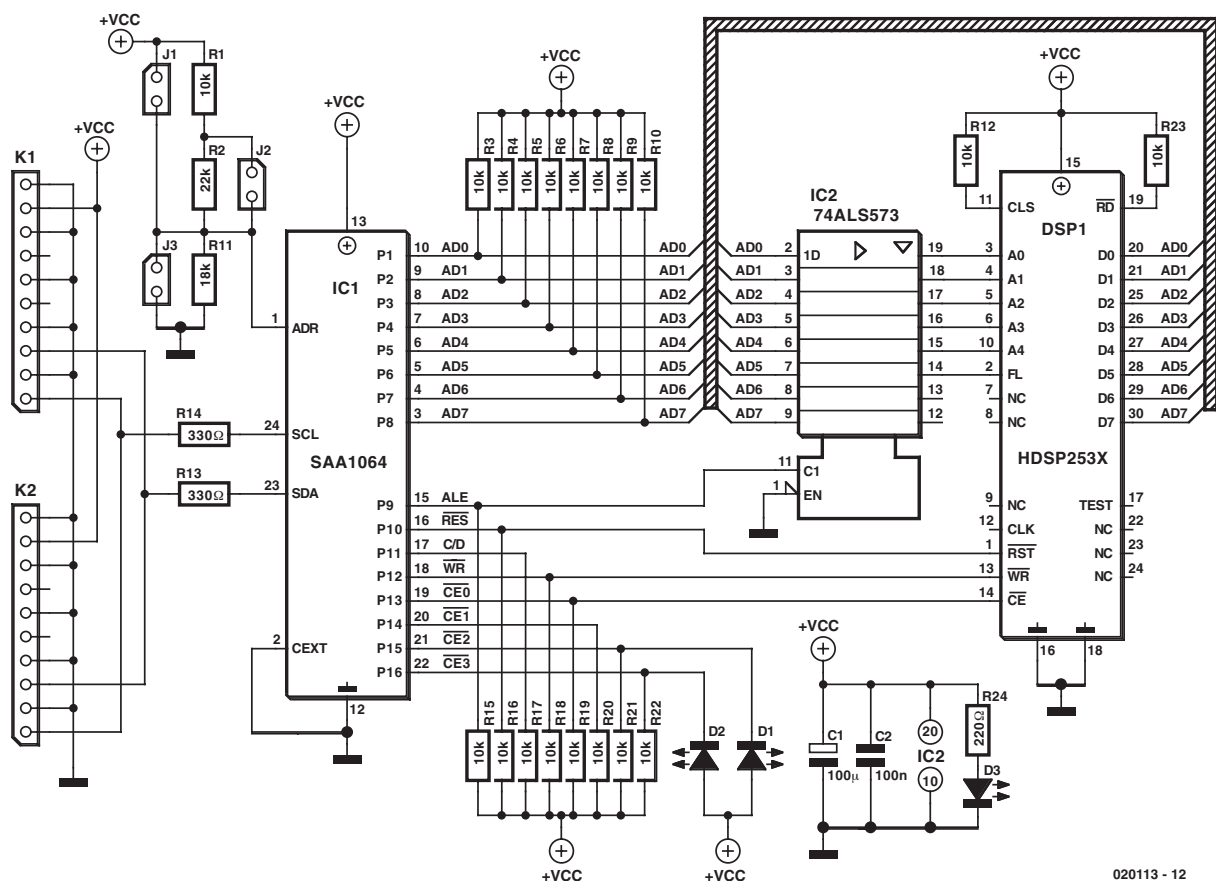


Figure 2. Circuit diagram of the I²C driver with HDSP2533 display.

– ALE – Address Latch Enable

This pin receives a pulse that latches the Address bits on the falling edge.

– $\overline{\text{Reset}}$

The active low Reset signal may be used to reset slave devices.

– $\text{C}/\overline{\text{D}}$ – Command / Data (Not)

This control signal may be used to select Command or Data registers on a slave device. LCDs based on HD44780 driver chip would refer to this signal as RS (Register Select).

– $\overline{\text{WR}}$ – Write

The active low $\overline{\text{WR}}$ signal activates the write operation on the slave device. Some devices, like the HD44780, may require an active high 'enable' signal instead.

– $\overline{\text{CE0}} \dots \overline{\text{CE3}}$ – Chip Enable

Four general-purpose control bits are available. They may be used to select a specific peripheral device when more than one is connected, in this case only one of these $\overline{\text{CE}}$ signals should be active at any time.

Note that there no $\overline{\text{RD}}$ signal is provided since the SAA1064 does not support reading back data from a slave. Please make sure that any $\overline{\text{RD}}$ control signals on slave devices are **disabled** (i.e., pulled high) to prevent shorting the slave databus outputs with the SAA1064 Port 1 Address/Databus!

All Port outputs need a 10-k Ω pull-up (R3-R10 and R15-R22) because the outputs are 'open-collector' current sources. The resistors provide defined logic levels for the slave devices.

HDSP253x Example

The Hewlett-Packard (now known as 'Agilent') HDSP253x device is an 8-digit 'Smart Alphanumeric Display' available in several colours (green, orange, red and yellow). The on-board decoder accepts 128 different ASCII symbols. Character codes are stored in internal RAM and shown on the display without external refresh action. The device can show flashing characters, has seven brightness levels, several blanking functions and supports 16 user defined characters (UDCs) to generate special symbols — See **Figure 3**.

Similar displays are available from Siemens (e.g. PD2435), some having only four characters per device (e.g. SLR2016). See also [4] and [5]. These displays differ from the HDSP in size, number of characters and other details. The example software and hardware will therefore need some modification when replacing the HDSP with the Siemens

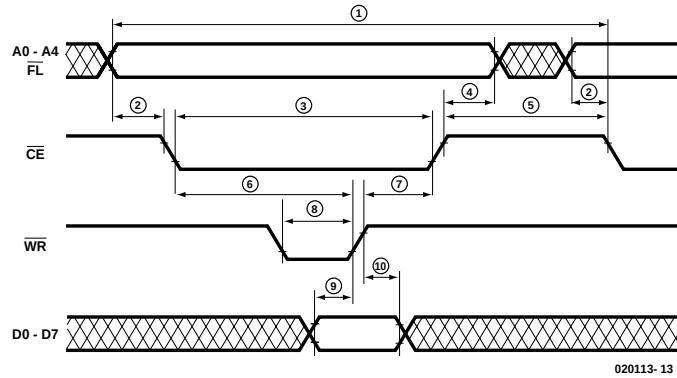


Figure 3. Timing diagram for HDSP253x.

devices, but the basic operation is identical.

The timing diagram of the HDSP data-, address- and control-lines is shown in **Figure 4**. The Address bus selects the digit position, the Databus provides the ASCII character code and $\overline{\text{WR}}$ will activate the data transfer to the display. Please note that the $\overline{\text{CE}}$ pin of the device may not be tied low permanently but must be toggled between each Read or Write operation.

Hooking up the HDSP253x to the General Purpose Driver Interface is straightforward: the Data bus of the HDSP connects to AD0-AD7 of SAA1064 Port 1, Address lines A0-A4 are connected to the corresponding Address lines on latch U2. For simplicity, the $\overline{\text{FL}}$ control line is connected to A5 of the latch. The $\overline{\text{FL}}$ line allows access to the 'flashing character' RAM of the HDSP. The remaining control lines ($\overline{\text{RST}}$, $\overline{\text{WR}}$ and $\overline{\text{CE0}}$) are connected to Port 2 of the SAA1064. Two control pins on the HDSP are pulled 'high': the $\overline{\text{RD}}$ line (to avoid bus collision) and the CLS pin (to select the internal clock of the display).

In this application example, the spare CE2 and CE3 drive two LEDs (D1, D2) that provide additional signalling. Both LEDs can be connected to the SAA1064 without additional series resistors because the outputs of the SAA1064 are in fact current sources.

Software

The software for the Interface may be downloaded from this month's Free Downloads items at www.elek-tor-electronics.co.uk

[tor-electronics.co.uk](http://www.elek-tor-electronics.co.uk). The program was developed in 8051 assembler (Metalink). The original Intel 8051 does not have I²C hardware and the necessary routines were implemented through 'bit banging' of two port pins representing SDA and SCL. Supporting routines for I²C and serial port communication, utility sub-routines and specific control software for the slave device are all located in separate modules. This allows easier software maintenance and reuse. All modules have the same general structure and consist of two parts:

- Mod_cnst.inc, contains constant definitions and variable declarations;
- Mod_sub.asm, contains the sub-routines and program memory constants (e.g. tables).

The software modules will be briefly described below. Details can be found in the source code that is provided with comments and explanations.

SCL and SDA may be assigned to any of the available processor port pins by setting an 'EQU' in the source code of 'I2C_cnst.inc'. Separate macro's are used for I²C bit Setup and bit Hold times to allow easier porting of the software to faster 8051 types (e.g. Atmel versions running at 24 MHz or higher). The default settings are adequate for an 8051/80C535 running at 11.0592 MHz. No particular delays are required in that case because the software is now slower than the maximum allowed I²C bus speed.

The lowest level I²C software generates the I²C 'Start' and 'Stop'

sequences. The I²C bus is initialised to the appropriate levels after a 'power-on' by calling 'I2C_init'. The second level of the I²C code provides Byte Send and Receive functionality. The highest level of I²C code sends or receives an I²C message. This involves generating I²C Start and Stop conditions and sending a Slave Address and an optional I²C sub address. Flags are used to set or receive the I²C Acknowledge bit.

The general software for the SAA1064 interface provides subroutines to set values on the 'Address/Databus' and on the 'Control bus'. There is also a routine that allows writing an Address to the Address latch, which consists of first writing data to the Address/Databus and then toggling ALE. Separate macros are provided to allow setting or resetting a specific Control bus bit. The SAA1064 hardware inverts all actual signal levels: setting a port bit 'high' will result in a 'low' level on the port (due to the current source output circuit). The inverting effect is compensated in the low-level routines just before I²C data is transmitted to the SAA1064.

Based on the general SAA1064 routines, a set of specific software modules is available to control the Hewlett-Packard display (files: 'hpd_cnst.inc' and 'hpd_sub.asm'). The Low level 'I2C_HPDIInit' routine initialise the display device. This routine first selects the correct mode for the SAA1064 (e.g. no multiplexing) then initialises levels for all control bits (e.g. ALE is low), generates a Reset sequence for the HDSP and finally programs the control registers of the display and clears the display memory. Higher level routines are

available that write characters (I2C_HPDPutChar) and strings (I2C_HPDPutStrCnst and I2C_HPDPutStrBuf). The display cursor will automatically wrap around. The cursor may be set by the 'I2C_HPDPGoTo' routine. Additional support functions allow uploading of a user-defined pattern (5*7 bits) to display special symbols or logo's (I2C_HPDUUDC).

The serial port software (contained in 'ser_cnst.inc' and 'ser_sub.asm') provides initialisation of the 8051/80535 serial port hardware. Assembler directives are available to select 4800/9600 baud and select the CPU clock frequency (11.0592 or 12 MHz). Basic routines allow character transmit and receive and string transmission to a terminal program (e.g., Hyperterminal).

The UTL module ('utl_cnst.inc' and 'utl_sub.asm') provides general support functions e.g. ASCII to Hex, translation to upper case etc. This module also has special character/symbol definitions that may be uploaded into the HDSP device. Examples are a battery full/empty symbol and some 'pac-man' icons.

The main software module is 'Test51.asm' and it contains the actual application, it uses the #include assembler directive to incorporate all the necessary support modules.

Default settings for the 'Test51' software are: 80C535 processor at 12 MHz, 9600 baud serial port, SDA at P3.4, SCL at P1.0, HDSP device at I²C slave address 76h (jumper SW1 placed). Selection of a specific processor type (8051 or 80C535) and other modes/options (e.g. bau-

date) are possible by activating assembler directives in 'Test51' or one of its #included modules.

The 'Test51' software initialises the 8051/80C535 (serial port, stack, etc.), the I²C hardware (SDA and SCL pins, flags, etc.) and the I²C general purpose interface (i.e., the SAA1064). Finally the slave display device connected to the interface is initialised (e.g., clear the display, reset the cursor, etc.). After completing the initialisation process the 'Test51' software enters an infinite loop that activates the HDSP display at specific times to show different strings. The operation of the software may be observed by connecting a terminal program (e.g., hyper terminal) to the serial port. A text string is displayed after initialisation and a '#' character is transmitted for each iteration of the main loop.

Experiments

Readers can modify the example hardware and code to meet their own needs. The structured software is easy to understand and for most devices software adaptation should be straightforward. Some hints :

- Remember that the I²C 'bit-banging' software is relatively slow and any required delays (e.g. data setup times before activating WR) are normally satisfied without additional delays.
- Applications that really need read-back of data (e.g. graphics display where a single bit in the display RAM is modified) can sometimes still be supported by keeping a 'copy' of the appropriate data in the processor's RAM, modify the particular value of the copy and write back the entire result to the slave.

(020113-1)

The software for this project may be downloaded as file number **020113-11.zip** from the Free Downloads section at www.elektor-electronics.co.uk, under month of publication.

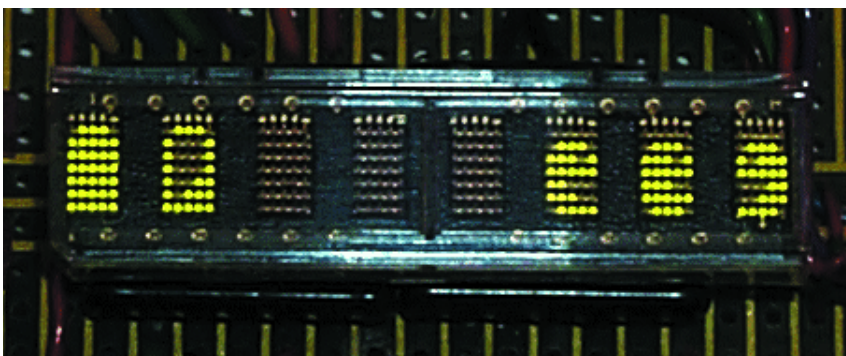


Figure 4. Examples of User Defined Characters.

References:

- [1] I²C Alphanumeric Display, Elektor Electronics September 1993.
- [2] I²C LED Display, Elektor Electronics June 1992.
- [3] www.agilent.com
- [4] PC Dot Matrix Display, Elektor Electronics November 2001.
- [5] Distance Measurement using Infrared, Elektor Electronics July/August 2002.